Line attractor dynamics in recurrent networks for sentiment classication

Niru Maheswaranathan [*1] **Alex H. Williams** [*2 1] **Matthew D. Golub** [2] **Surya Ganguli** [2 1] **David Sussillo** [1]

Abstract

Recurrent neural networks (RNNs) are a powerful tool for modeling sequential data. Despite their widespread usage, understanding how RNNs solve complex problems remains elusive. Here, we characterize how popular off-the-shelf architectures (including LSTMs, GRUs, and vanilla RNNs) perform document-level sentiment classification. Despite their theoretical capacity to implement complex, high-dimensional computations, we find that all architectures converge to highly interpretable, low-dimensional representations. We identify a simple mechanism, integration along an approximate line attractor, and find this mechanism present across RNN architectures (including LSTMs, GRUs, and vanilla RNNs). Overall, these results demonstrate that surprisingly universal and human interpretable computations can arise across a range of RNNs.

1. Introduction

Recurrent neural networks (RNNs) are a popular tool for sequence modelling tasks. These architectures are thought to learn complex relationships in input sequences, and exploit this structure in a nonlinear fashion. RNNs are typically viewed as black boxes, despite considerable interest in better understanding how they function.

Here, we focus on studying how recurrent networks solve document-level sentiment analysis—a simple, but longstanding benchmark task for language modeling (Liu, 2015; Zhang et al., 2018). We demonstrate that popular RNN architectures, despite having the capacity to implement high-dimensional and nonlinear computations, in practice converge to low-dimensional representations when trained against this task. Moreover, using analysis techniques from dynamical systems theory, we show that locally linear approximations to the nonlinear RNN dynamics are highly interpretable. In particular, they all involve approximate low-dimensional line attractor dynamics–a useful dynamical feature that can be implemented by linear dynamics and used to store an analog value (Seung, 1996). Furthermore, we show that this mechanism is surprisingly consistent across a range of RNN architectures.

[*]Equal contribution [1]Google Brain, Google Inc., Mountain View, CA [2]Neurosciences Program, Stanford University, Stanford, CA. Correspondence to: David Sussillo <sussillo@google.com>.

2. Methods

2.1. Training

We trained four RNN architectures–LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), Update Gate RNN (UGRNN) (Collins et al., 2016), and standard (vanilla) RNNs–on binary sentiment classifcation tasks. We trained each network type on each of three datasets: the IMDB movie review dataset, which contains 50,000 highly polarized reviews (Maas et al., 2011); the Yelp review dataset, which contained 500,000 user reviews (Zhang et al., 2015); and the Stanford Sentiment Treebank, which contains 11,855 sentences taken from movie reviews (Socher et al., 2013). For each task and architecture, we analyzed the best performing networks, selected using a validation set (see Appendix A for details).

2.2. Fixed point analysis

We analyzed trained networks by linearizing the dynamics around approximate fixed points. Approximate fixed points are state vectors $\{\mathbf{h}_1^*, \mathbf{h}_2^*, \mathbf{h}_3^*, \cdots\}$ which stay the same after applying the RNN, that is, $\mathbf{h}_i^* \approx F(\mathbf{h}_i^*, \mathbf{x}{=}\mathbf{0})$ (Sussillo & Barak, 2013). Briefly, we find these fixed points numerically by first defining a loss function $q = \frac{1}{N}\|\mathbf{h} - F(\mathbf{h}, \mathbf{0})\|_2^2$, and then minimizing q with respect to hidden states, $\mathbf{h}$, using standard auto-differentiation methods (Golub & Sussillo, 2018). We ran this optimization multiple times starting from different initial values of $\mathbf{h}$. These initial conditions were sampled randomly from the state activation visited by the trained network, which was done to intentionally sample states related to the operation of the RNN.

3. Results

For brevity, we explain our approach using the working example of the LSTM trained on the Yelp dataset (Figs. 1-3). We find similar results for all architectures and datasets, these are shown in Appendix B.

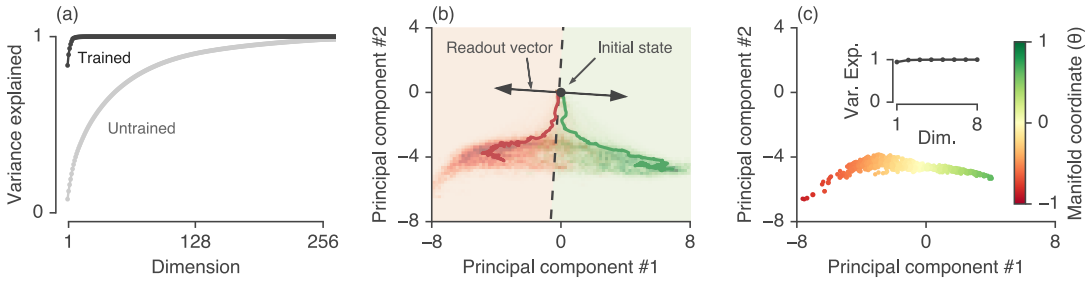

Figure 1. LSTMs trained to identify the sentiment of Yelp reviews explore a low-dimensional volume of state space. (a) PCA on LSTM hidden states - PCA applied to all hidden states visited during 1000 test examples for untrained (light gray) vs. trained (black) LSTMs. After training, most of the variance in LSTM hidden unit activity is captured by a few dimensions. (b) RNN state space - Projection of LSTM hidden unit activity onto the top two principal components (PCs). 2D histogram shows density of visited states for test examples colored for negative (red) and positive (green) documents. Two example trajectories are shown for a document of each type (red and green solid lines, respectively). The projection of the initial state (black dot) and readout vector (black arrows) in this low-dimensional space are also shown. Dashed black line shows a readout value of 0. (c) Approximate fixed points - Projection of approximate fixed points of the LSTM dynamics (see Methods) onto the top PCs. The fixed points lie along a 1-D manifold (inset shows variance explained by PCA on the approximate fixed points), parameterized by a coordinate θ (see Methods).

3.1. RNN dynamics are low-dimensional

As an initial exploratory analysis step, we performed principal components analysis (PCA) on the RNN states concatenated across 1,000 test examples. The top 2-3 PCs explained $\sim$90% of the variance in hidden state activity (Fig. 1a, black line). The distribution of hidden states visited by untrained networks on the same test set was much higher dimensional (Fig. 1a, gray line), suggesting that training the networks stretched the geometry of their representations along a low-dimensional subspace.

We then visualized the RNN dynamics in this low-dimensional space by forming a 2D histogram of the density of RNN states colored by the sentiment label (Fig. 1b). This visualization is reminiscent of integration dynamics along a line attractor–a well-studied mechanism for evidence accumulation in simple recurrent networks (Seung, 1996; Mante et al., 2013)–and we reasoned that similar dynamics may be used for sentiment classification.

The hypothesis that RNNs approximate line attractor dynamics during sentiment classification makes four specific predictions, which we investigate and confirm in subsequent sections. First, the fixed points form an approximately 1D manifold that is aligned/correlated with the readout weights (Section 3.2). Second, all fixed points are attracting and marginally stable. That is, in the absence of input (or, perhaps, if a string of neutral/uninformative words are encountered) the RNN state should rapidly converge to the closest fixed point and then should not change appreciably (Section 3.4). Third, locally around each fixed point, inputs representing positive vs. negative evidence should produce linearly separable effects on the RNN state vector along some dimension (Section 3.5). Finally, these instantaneous effects should be integrated by the recurrent dynamics along the direction of the 1D fixed point manifold (Section 3.5).

3.2. RNNs follow a 1D manifold of stable fixed points

We numerically identified the location of $\sim$500 RNN fixed points using previously established methods (Sussillo & Barak, 2013; Golub & Sussillo, 2018). We then projected these fixed points into the same low-dimensional space used in Fig. 1b. Although the PCA projection was fit to the RNN hidden states, and not the fixed points, a very high percentage of variance in fixed points was captured by this projection (Fig. 1c, inset), suggesting that that the RNN states remain close to the manifold of fixed points. We call the vector that describes the main axis of variation of the 1D manifold $\mathbf{m}$. Consistent with the line attractor hypothesis, the fixed points appeared to be spread along a 1D curve when visualized in PC space, and furthermore the principal direction of this curve was aligned with the readout weights (Fig. 1c). We further verified that this low-dimensional approximation was accurate by using locally linear embedding (LLE; Roweis & Saul 2000) to parameterize a 1D manifold of fixed points in the raw, high-dimensional data. This provided a scalar coordinate, $\theta_i \in [-1, 1]$, for each fixed point, which was well-matched to the position of the fixed point manifold in PC space (coloring of points in Fig. 1c).

3.3. Linear approximations of RNN dynamics

We next aimed to demonstrate that the identified fixed points were marginally stable, and thus could be used to preserve accumulated information from the inputs. To do this, we used a standard linearization procedure (Khalil, 2001) to obtain an approximate, but highly interpretable, description of the RNN dynamics near the fixed point manifold. Given the last state $\mathbf{h}_{t-1}$ and the current input $\mathbf{x}_t$, the approach

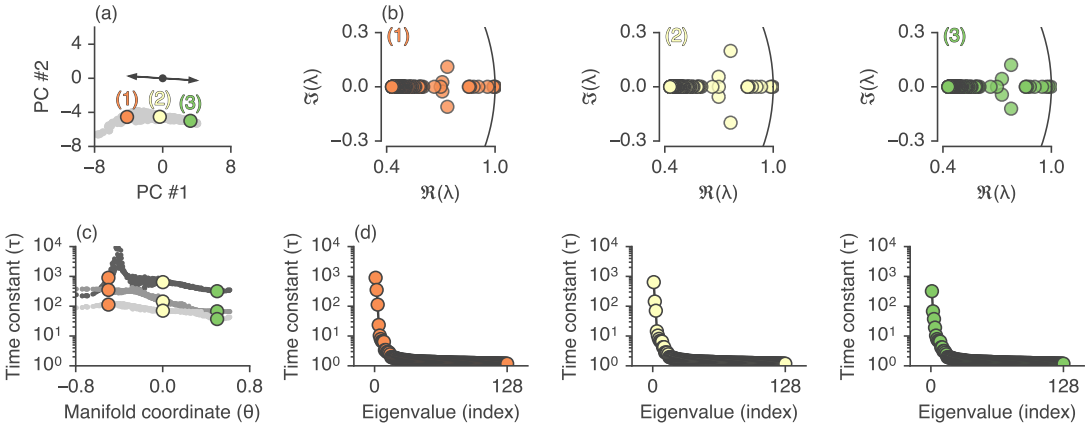

Figure 2. Characterizing the top eigenmodes of each fixed point. (a) Same plot as in Fig. 1c (fixed points are grey), with three example fixed points highlighted. (b) For each of these fixed points, we compute the LSTM Jacobian (see Methods) and show the distribution of eigenvalues (colored circles) in the complex plane (black line is the unit circle). (c-d) The time constants (τ in terms of # of input tokens) associated with the eigenvalues. (c) The time constant for the top three modes for all fixed points as function of the manifold coordinate (θ). (d) All time constants for all eigenvalues associated with the three highlighted fixed points.

is to locally approximate the update rule with a first-order Taylor expansion:

$$\begin{aligned}\mathbf{h}_t &= F(\mathbf{h}^* + \Delta\mathbf{h}_{t-1}, \mathbf{x}^* + \Delta\mathbf{x}_t) \\ &\approx F(\mathbf{h}^*, \mathbf{x}^*) + \mathbf{J}^{\text{rec}}\Delta\mathbf{h}_{t-1} + \mathbf{J}^{\text{inp}}\Delta\mathbf{x}_t \end{aligned} \quad (1)$$

where $\Delta\mathbf{h}_{t-1} = \mathbf{h}_{t-1} - \mathbf{h}^*$ and $\Delta\mathbf{x}_t = \mathbf{x}_t - \mathbf{x}^*$, and $\{\mathbf{J}^{\text{rec}}, \mathbf{J}^{\text{inp}}\}$ are Jacobian matrices of the system: $J^{\text{rec}}_{ij}(\mathbf{h}^*, \mathbf{x}^*) = \frac{\partial F(\mathbf{h}^*, \mathbf{x}^*)_i}{\partial h^*_j}$ and $J^{\text{inp}}_{ij}(\mathbf{h}^*, \mathbf{x}^*) = \frac{\partial F(\mathbf{h}^*, \mathbf{x}^*)_i}{\partial x^*_j}$. We choose $\mathbf{h}^*$ to be a numerically identified fixed point and $\mathbf{x}^* = \mathbf{0}$, thus we have $F(\mathbf{h}^*, \mathbf{x}^*) \approx \mathbf{h}^*$ and $\Delta\mathbf{x}_t = \mathbf{x}_t$. Under this choice, equation (1) reduces to a discrete-time linear dynamical system:

$$\Delta\mathbf{h}_t = \mathbf{J}^{\text{rec}}\Delta\mathbf{h}_{t-1} + \mathbf{J}^{\text{inp}}\mathbf{x}_t. \quad (2)$$

It is important to note that both Jacobians depend on which fixed point we choose to linearize around, and should thus be thought of as functions of $\mathbf{h}^*$; for notational simplicity we do not denote this dependence explicitly.

By reducing the nonlinear RNN to a linear system, we can analytically estimate the network's response to a sequence of T inputs. In this approximation, the effect of each input $\mathbf{x}_t$ is decoupled from all others; that is, the final state is given by the sum of all individual effects.[1] We can restrict our focus to the effect of a single input, $\mathbf{x}_t$ (i.e. a single term in this sum). Let $k = T - t$ be the number of time steps between $\mathbf{x}_t$ and the end of the document. The total effect of $\mathbf{x}_t$ on the final RNN state becomes:

$$\mathbf{R}\mathbf{\Lambda}^k\mathbf{L}\mathbf{J}^{\text{inp}}\mathbf{x}_t = \sum_{a=1}^{N} \lambda_a^k \mathbf{r}_a \boldsymbol{\ell}_a^\top \mathbf{J}^{\text{inp}}\mathbf{x}_t, \quad (3)$$

[1] We consider the case where the network has closely converged to a fixed point, so that $\mathbf{h}_0 = \mathbf{h}^*$ and thus $\Delta\mathbf{h}_0 = \mathbf{0}$.

where $\mathbf{L} = \mathbf{R}^{-1}$, the columns of $\mathbf{R}$ (denoted $\mathbf{r}_a$) contain the *right eigenvectors* of $\mathbf{J}^{\text{rec}}$, the rows of $\mathbf{L}$ (denoted $\boldsymbol{\ell}_a^\top$) contain the *left eigenvectors* of $\mathbf{J}^{\text{rec}}$, and $\mathbf{\Lambda}$ is a diagonal matrix containing complex-valued eigenvalues, $\lambda_1 > \lambda_2 > \ldots > \lambda_N$, which are sorted based on their magnitude.

3.4. An analysis of integration time constants.

From equation 3 we see that $\mathbf{x}_t$ affects the representation of the network through N terms (called the *eigenmodes* or *modes* of the system). The magnitude of each mode after k steps is given by the λ_a^k; thus, the size of each mode either reduces to zero or diverges exponentially fast, with a time constant given by: $\tau_a = \left|\frac{1}{\log(|\lambda_a|)}\right|$. This time constant has units of tokens (or, roughly, words) and yields an interpretable number for the effective memory of the system. Fig. 2 plots the eigenvalues and associated time constants and shows the distribution of all eigenvalues at three representative fixed points along the fixed point manifold (Fig. 2a). In Fig. 2c, we plot the decay time constant of the top three modes; the slowest decaying mode persists after $\sim$1000 time steps, while the next two modes persists after $\sim$100 time steps, with lower modes decaying even faster. Since the average review length for the Yelp dataset is $\sim$175 words, only a small number of modes could represent information from the beginning of the document.

Overall, these eigenvalue spectra are consistent with our observation that RNN states only explore a low-dimensional subspace when performing sentiment classification. RNN activity along the majority of dimensions is associated with fast time constants and is therefore quickly forgotten.

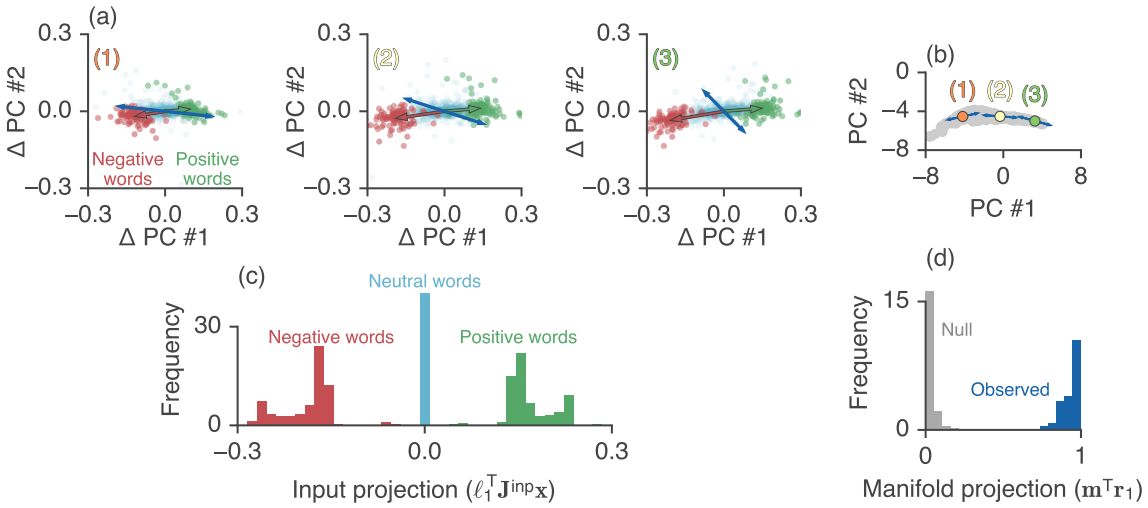

Figure 3. Effect of different word inputs from the Yelp corpus on the LSTM state vector. (a) Instantaneous effect of word inputs, $\mathbf{J}^{\text{inp}}\mathbf{x}$, for positive, negative, and neutral words (green, red, cyan dots). The green and red arrows point to the center of mass for the positive and negative words, respectively. Blue arrows denote $\boldsymbol{\ell}_1$, the top *left* eigenvector. The PCA projection is the same as Fig. 1c, but centered around each fixed point. Each plot denotes a separate fixed point (labeled in panel b). (b) Same plot as in Fig. 1c, with three example fixed points highlighted (approximate fixed points in grey). Blue arrows denote $\mathbf{r}_1$, the top *right* eigenvector. In all cases $\mathbf{r}_1$ is aligned with the orientation of the manifold, $\mathbf{m}$, consistent with an approximate line attractor. (c) Average of $\boldsymbol{\ell}_1^\top \mathbf{J}^{\text{inp}}\mathbf{x}$ over 100 different words, shown for positive, negative, neutral words. Histogram displays the distribution of this input projection over all fixed points. (d) Distribution of $\mathbf{r}_1^\top \mathbf{m}$ (overlap of the top right eigenvector with the fixed point manifold) over all fixed points. Null distribution is randomly generated unit vectors of the size of the hidden state.

3.5. Left and right eigenvectors

Restricting our focus to the top eigenmode for simplicity (there may be a few slow modes of integration), the effect of a single input, $\mathbf{x}_t$, on the network activity (equation 3) becomes: $\mathbf{r}_1 \boldsymbol{\ell}_1^\top \mathbf{J}^{\text{inp}}\mathbf{x}$, where we have dropped the dependence on t since $\lambda_1 \approx 1$, so the effect of $\mathbf{x}$ is largely insensitive to the exact time it was input to system. Using this expression, we separately analyzed the effects of specific words.

We first examined the term $\mathbf{J}^{\text{inp}}\mathbf{x}$ for various choices of $\mathbf{x}$ (i.e. various word tokens). This quantity represents the instantaneous linear effect of $\mathbf{x}$ on the RNN state vector and is shared across all eigenmodes. We projected the resulting vectors onto the same low-dimensional subspace shown in Fig. 1c. We see that positive and negative valence words push the hidden state in opposite directions. Neutral words, in contrast, exert much smaller effects on the RNN state (Fig. 3).

While $\mathbf{J}^{\text{inp}}\mathbf{x}$ represents the instantaneous effect of a word, only the features of this input that overlap with the top few eigenmodes are reliably remembered by the network. The scalar quantity $\boldsymbol{\ell}_1^\top \mathbf{J}^{\text{inp}}\mathbf{x}$, which we call the *input projection*, captures the magnitude of change induced by $\mathbf{x}$ along the eigenmode associated with the longest timescale. Again we observe that the valence of $\mathbf{x}$ strongly correlates with this quantity: neutral words have an input projection near zero while positive and negative words produced larger magnitude responses of opposite sign. Furthermore, this is reliably observed across all fixed points. Fig. 3c shows the average input projection for positive, negative, and neutral words; the histogram shows the distribution of these average effects across all fixed points along the line attractor.

Finally, if the input projection onto the top eigenmode is non-negligible, then the right eigenvector $\mathbf{r}_1$ (which is normalized to unit length) represents the direction along which $\mathbf{x}$ is integrated. If the RNN implements an approximate line attractor, then $\mathbf{r}_1$ (and potentially other slow modes) should align with the principal direction of the manifold of fixed points, $\mathbf{m}$. We indeed observe a high degree of overlap between $\mathbf{r}_1$ and $\mathbf{m}$ both visually in PC space (Fig. 3b) and quantitatively across all fixed points (Fig. 3d).

4. Discussion

In this work we applied dynamical systems analysis to understand how RNNs solve sentiment analysis. We found a simple mechanismintegration along a line attractorpresent in multiple architectures trained to solve the task. Overall, this work provides preliminary, but optimistic, evidence that different, highly intricate network models can converge to similar solutions that may be reduced and understood by human practitioners.

References

Cho, K., Merrienboer, B. v., Gulcehre, C., Bougares, F., Schwenk, H., and Bengio, Y. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proc. Conference on Empirical Methods in Natural Language Processing*, Unknown, Unknown Region, 2014.

Collins, J., Sohl-Dickstein, J., and Sussillo, D. Capacity and trainability in recurrent neural networks. *arXiv preprint arXiv:1611.09913*, 2016.

Golub, M. and Sussillo, D. FixedPointFinder: A tensorflow toolbox for identifying and characterizing fixed points in recurrent neural networks. *Journal of Open Source Software*, 3(31):1003, 2018. doi: 10.21105/joss.01003.

Hochreiter, S. and Schmidhuber, J. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Khalil, H. K. *Nonlinear Systems*. Pearson, 2001. ISBN 0130673897.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Liu, B. *Sentiment analysis: Mining opinions, sentiments, and emotions*. Cambridge University Press, 2015.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1*, HLT '11, pp. 142–150, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics. ISBN 978-1-932432-87-9.

Mante, V., Sussillo, D., Shenoy, K. V., and Newsome, W. T. Context-dependent computation by recurrent dynamics in prefrontal cortex. *Nature*, 503(7474):78, 2013.

Roweis, S. T. and Saul, L. K. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290 (5500):2323–2326, 2000. ISSN 0036-8075. doi: 10.1126/science.290.5500.2323.

Seung, H. S. How the brain keeps the eyes still. *Proceedings of the National Academy of Sciences*, 93(23): 13339–13344, 1996. ISSN 0027-8424. doi: 10.1073/pnas.93.23.13339.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Sussillo, D. and Barak, O. Opening the black box: low-dimensional dynamics in high-dimensional recurrent neural networks. *Neural computation*, 25(3):626–649, 2013.

Zhang, L., Wang, S., and Liu, B. Deep learning for sentiment analysis: A survey. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 8(4): e1253, 2018. doi: 10.1002/widm.1253.

Zhang, X., Zhao, J., and LeCun, Y. Character-level convolutional networks for text classification. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28*, pp. 649–657. Curran Associates, Inc., 2015.

A. Additional Methods

A.1. RNN architecture and notation

An RNN is defined by a nonlinear update rule $\mathbf{h}_t = F(\mathbf{h}_{t-1}, \mathbf{x}_t)$, which is applied recursively from an initial state $\mathbf{h}_0$ over a sequence of inputs $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_T$. Let N and M denote the dimensionality of the hidden states and the input vectors, so that $\mathbf{h}_t \in \mathbb{R}^N$ and $\mathbf{x}_t \in \mathbb{R}^M$. In sentiment classification, T represents the number of word tokens in a sequence, which can vary on a document-by-document basis. To process word sequences for a given dataset, we build a vocabulary and encode words as one-hot vectors. These are fed to a dense linear embedding layer with an embedding size of $M = 128$ ($\mathbf{x}_t$ are the embeddings in what follows). The word embeddings were trained from scratch simultaneously with the RNN. We considered four RNN architectures—LSTM (Hochreiter & Schmidhuber, 1997), GRU (Cho et al., 2014), UGRNN (Collins et al., 2016), and the vanilla RNN (VRNN)—each corresponding to a separate nonlinear update rule, $F(\cdot, \cdot)$. For the LSTM architecture, $\mathbf{h}_t$ consists of a concatenated hidden state vector and cell state vector so that N is twice the number of computational units; in all other architectures N is equal to the number of units. The RNN prediction is evaluated at the final time step T, and is given by $\hat{y} = \mathbf{w}^\top \mathbf{h}_T + b$, where we call $\mathbf{w} \in \mathbb{R}^N$ the *readout weights*. In the LSTM architecture, the cell state vector is not read out, and thus half of the entries in $\mathbf{w}$ are enforced to be zero under this notation.

	Bag of words	Vanilla RNN	Update Gate RNN	GRU	LSTM
Yelp 2015	93.37%	92.96%	95.67%	95.84%	95.05%
IMDB	88.53%	87.08%	87.96%	86.86%	86.93%
Stanford Sentiment	79.74%	78.09%	77.74%	80.25%	80.09%

Table 1. Test accuracies across all RNN architectures and datasets.

A.2. Datasets

We examined three benchmark datasets for sentiment classification: the IMDB movie review dataset, which contains 50,000 highly polarized reviews (Maas et al., 2011); the Yelp review dataset, which contained 500,000 user reviews (Zhang et al., 2015); and the Stanford Sentiment Treebank, which contains 11,855 sentences taken from movie reviews (Socher et al., 2013). The Stanford Sentiment Treebank also contains short phrases with labeled sentiments; these were not analyzed.

A.3. Training

For each task and architecture, we performed a randomized hyper-parameter search and selected the best networks based on a validation set. All models were trained using Adam (Kingma & Ba, 2014) with a batch size of 64. The hyper-parameter search was performed over the following ranges: the initial learning rate (10^{-5} to 10^{-1}), learning rate decay factor (0 to 1), gradient norm clipping (10^{-1} to 10), ℓ_2 regularization penalty (10^{-3} to 10^{-1}), the β_1 (0.5 to 0.99), and β_2 (0.9999 to 0.99) parameters of the Adam optimization routine. We additionally trained a bag-of-words model (logistic regression trained with word counts), as a baseline comparison. The accuracies of our final models on the held-out test set are summarized in Table 1. We analyzed the best performing models for each combination of architecture type and dataset.

A.4. Fixed Points

For each model, we numerically identified a large set of fixed points $\{\mathbf{h}_1^*, \mathbf{h}_2^*, \mathbf{h}_3^*, \cdots\}$ such that $\mathbf{h}_i^* \approx F(\mathbf{h}_i^*, \mathbf{x}{=}\mathbf{0})$ (Sussillo & Barak, 2013). Briefly, we accomplished this by first defining the loss function $q = \frac{1}{N}\|\mathbf{h} - F(\mathbf{h}, \mathbf{0})\|_2^2$, and then minimizing q with respect to hidden states, $\mathbf{h}$, using standard auto-differentiation methods (Golub & Sussillo, 2018). We ran this optimization multiple times starting from different initial values of $\mathbf{h}$. These initial conditions were sampled randomly from the state activation during the operation of the trained network, which was done to intentionally sample states related to the operation of the RNN. We varied the stopping tolerance for q using 9 points logarithmically spaced between 10^{-9} and 10^{-5} running the optimization from 1000 different initial conditions for each tolerance. This allowed us to find approximate fixed points of varying speeds. Values of q at numerical zero are true fixed points, while small but non-zero values are called slow points. Slow points are often reasonable places to perform a linearization, assuming that $\sqrt{q}$, which is akin to speed, is slow compared to the operation of the network.

B. Supplemental figures

Below we provide figures summarizing the linear integration mechanism for each combination of architecture (LSTMs, GRUs, Update Gate RNNs, Vanilla RNNs) and dataset (Yelp, IMDB, and Stanford Sentiment). Note that the first figure, LSTMs trained on Yelp, reproduces the figures in the main text–we include it here for completeness. The description of each panel is given in Figure 1, note that these descriptions are the same across all figures. We find that these mechanisms are remarkably consistent across architectures and datasets.

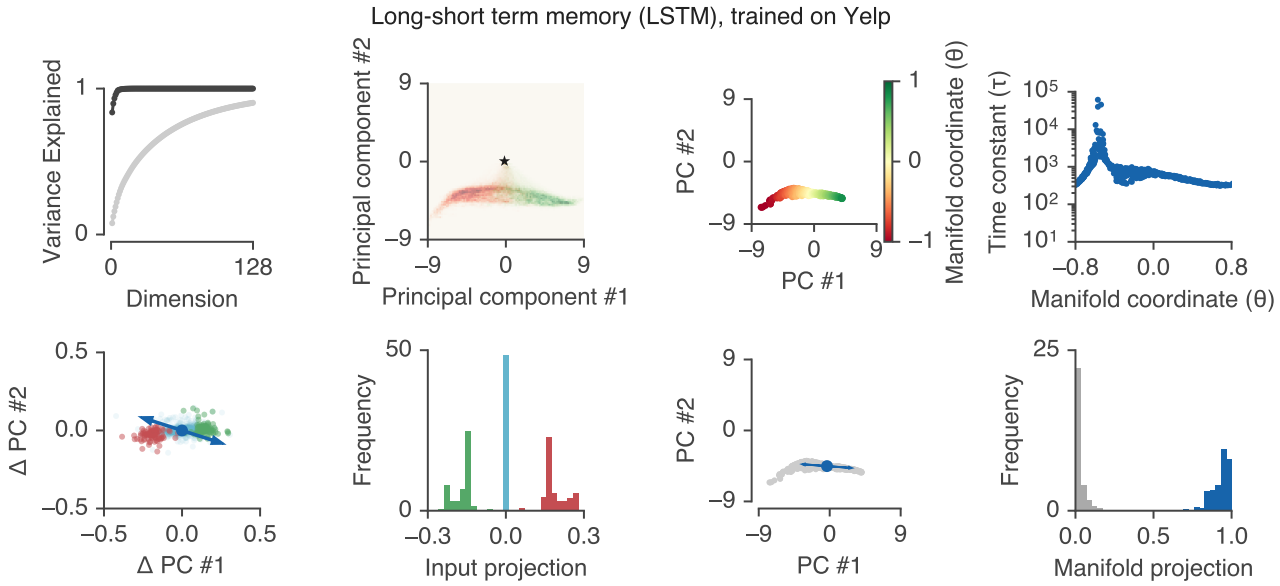

Figure 4. Summary plots for LSTM on Yelp reviews.
(upper left) PCA on RNN hidden states - PCA applied to all hidden states visited during 1000 test examples for untrained (light gray) vs. trained (black) LSTMs. After training, most of the variance in LSTM hidden unit activity is captured by a few dimensions.

(upper middle left) RNN state space - Projection of RNN hidden unit activity onto the top two principal components (PCs). 2D histogram shows density of visited states for test examples colored for negative (red) and positive (green) documents. Star indicates the initial hidden state.

(upper middle right) Approximate fixed points - Projection of approximate fixed points of the RNN dynamics (see Methods) onto the top PCs. The fixed points lie along a 1-D manifold, parameterized by a coordinate θ (see Methods).

(upper right) Time constant (τ) of memory as a function of position along the line attractor, θ.

(lower left) Instantaneous effect of word inputs, $\mathbf{J}^{\text{inp}}\mathbf{x}$, for positive (green), negative (red), and neutral (cyan) words. Blue arrows denote $\boldsymbol{\ell}_1$, the top *left* eigenvector. The PCA projection is the same as Fig. 2c, but centered around each fixed point.

(lower middle left) Average of $\boldsymbol{\ell}_1^\top \mathbf{J}^{\text{inp}}\mathbf{x}$ over 100 different words, shown for positive, negative, neutral words.

(lower middle right) Same plot as in Fig. 2c, with an example fixed point highlighted (approximate fixed points in grey). Blue arrows denote $\mathbf{r}_1$, the top *right* eigenvector.

(lower right) Distribution of $\mathbf{r}_1^\top \mathbf{m}$ (overlap of the top right eigenvector with the fixed point manifold) over all fixed points. Null distribution is randomly generated unit vectors of the size of the hidden state.

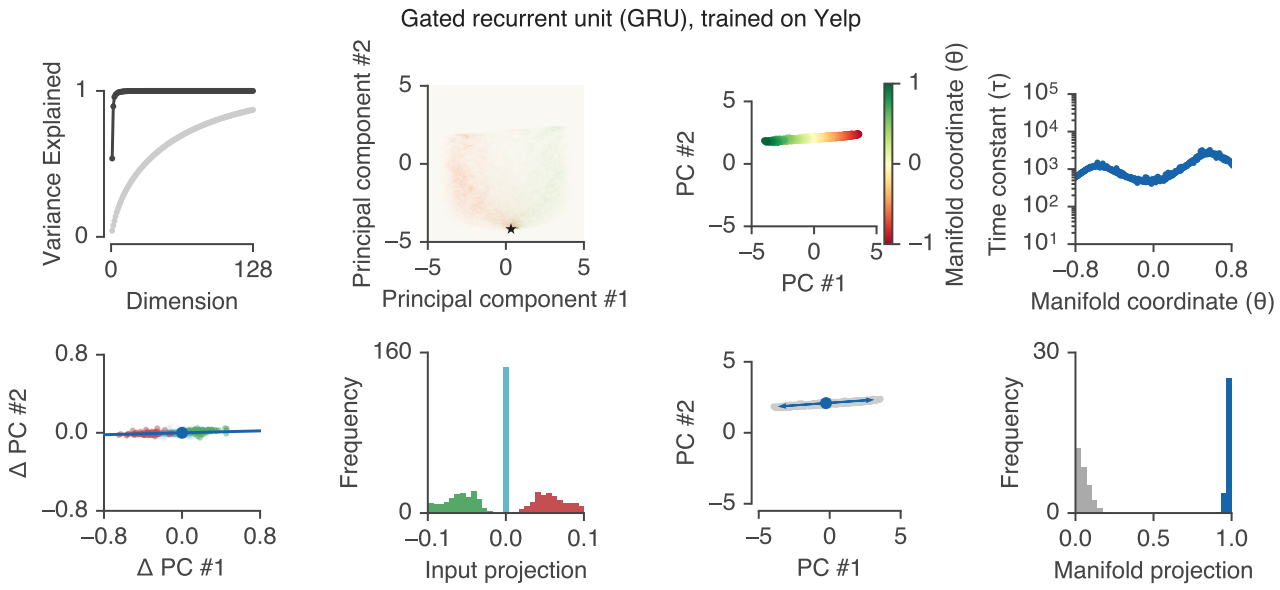

Figure 5. Summary plots for GRU on Yelp reviews. See first supplemental figure (LSTM on Yelp) for description.

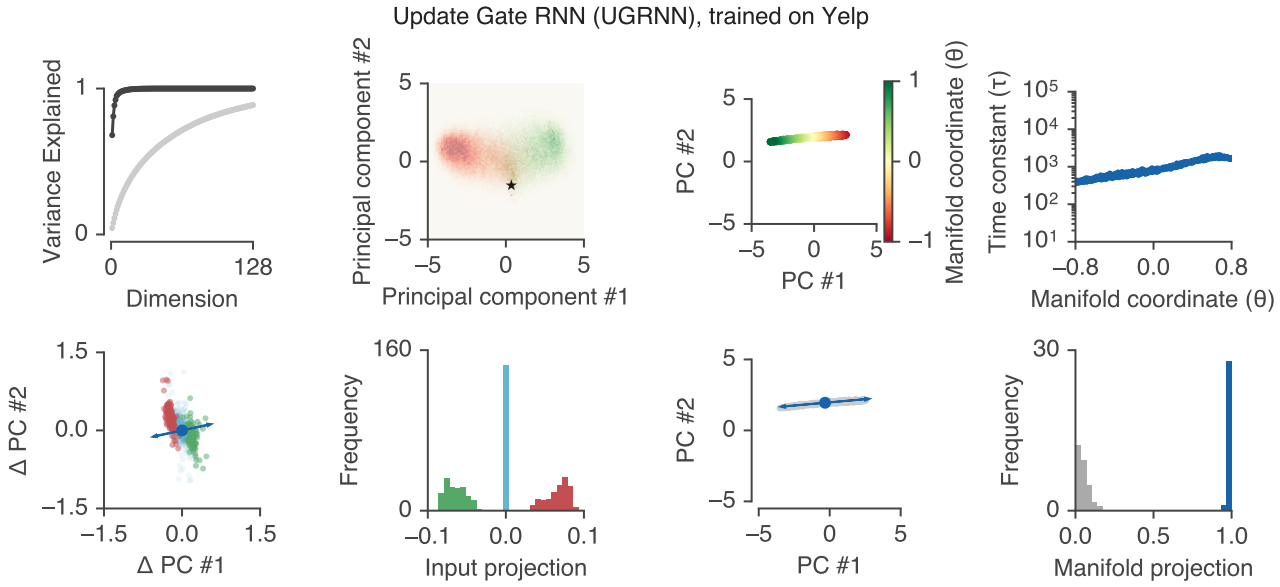

Figure 6. Summary plots for UGRNN on Yelp reviews. See first supplemental figure (LSTM on Yelp) for description.

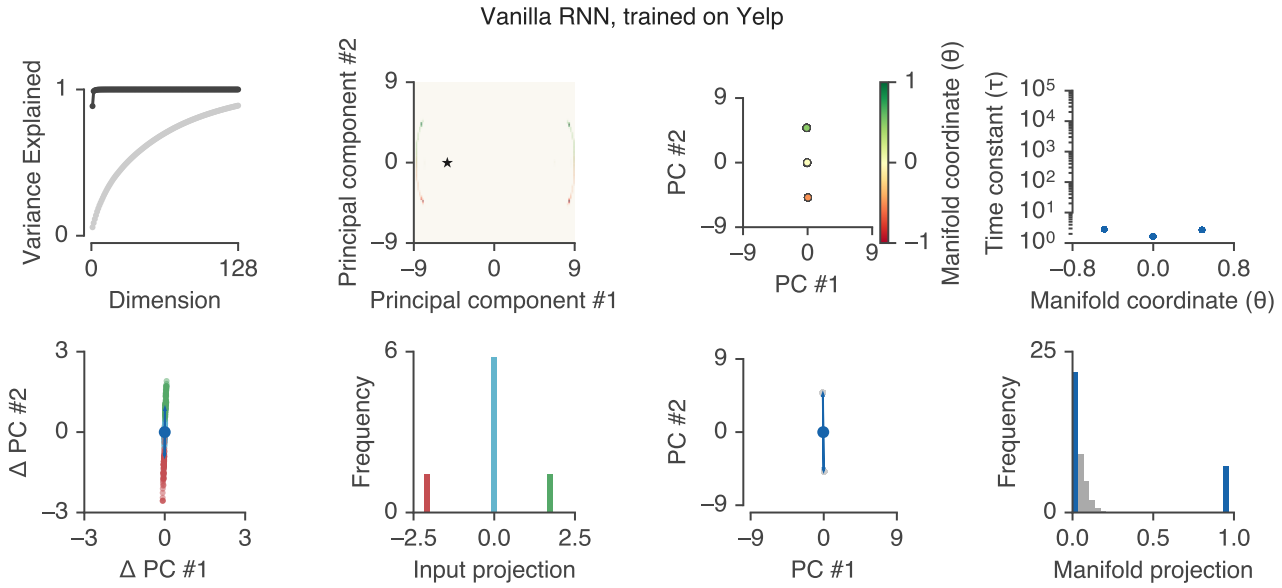

Figure 7. Summary plots for VRNN on Yelp reviews. See first supplemental figure (LSTM on Yelp) for description. Note this VRNN has an unstable oscillation as shown in distribution of hidden states in upper middle left PCA plot.

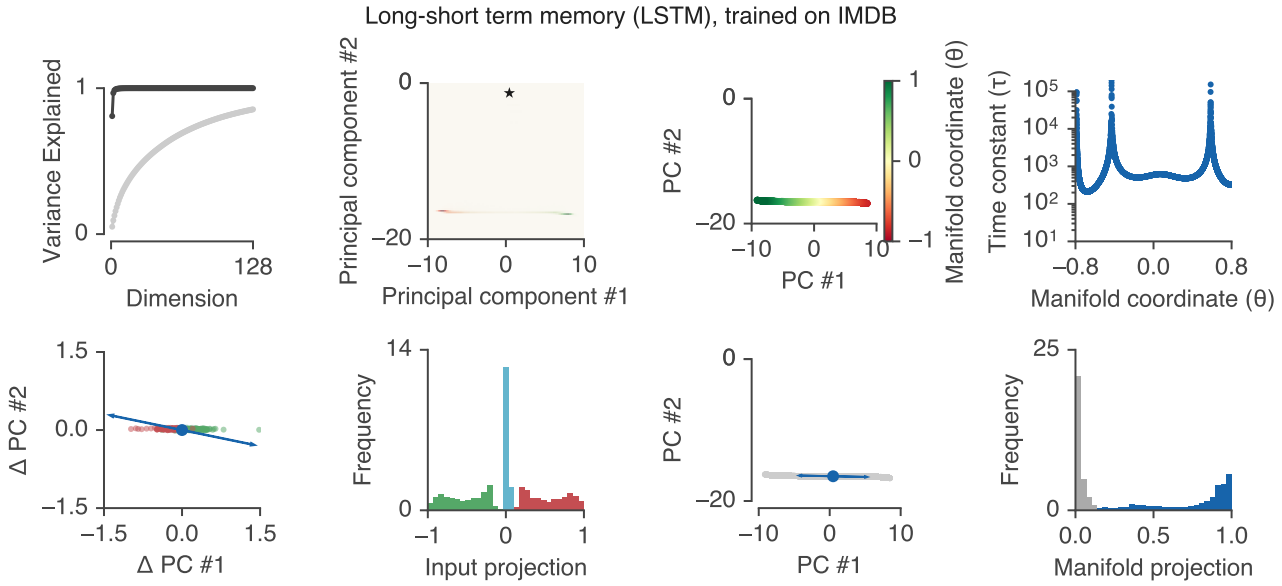

Figure 8. Summary plots for LSTM on IMDB reviews. See first supplemental figure (LSTM on Yelp) for description.

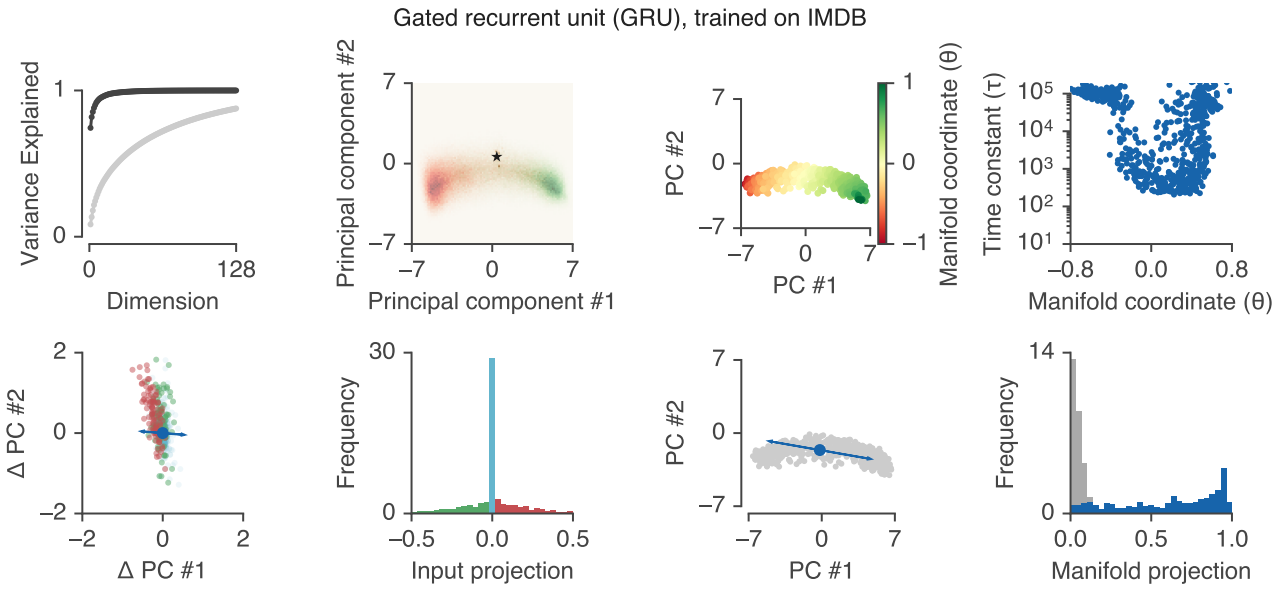

Figure 9. Summary plots for GRU on IMDB reviews. See first supplemental figure (LSTM on Yelp) for description.

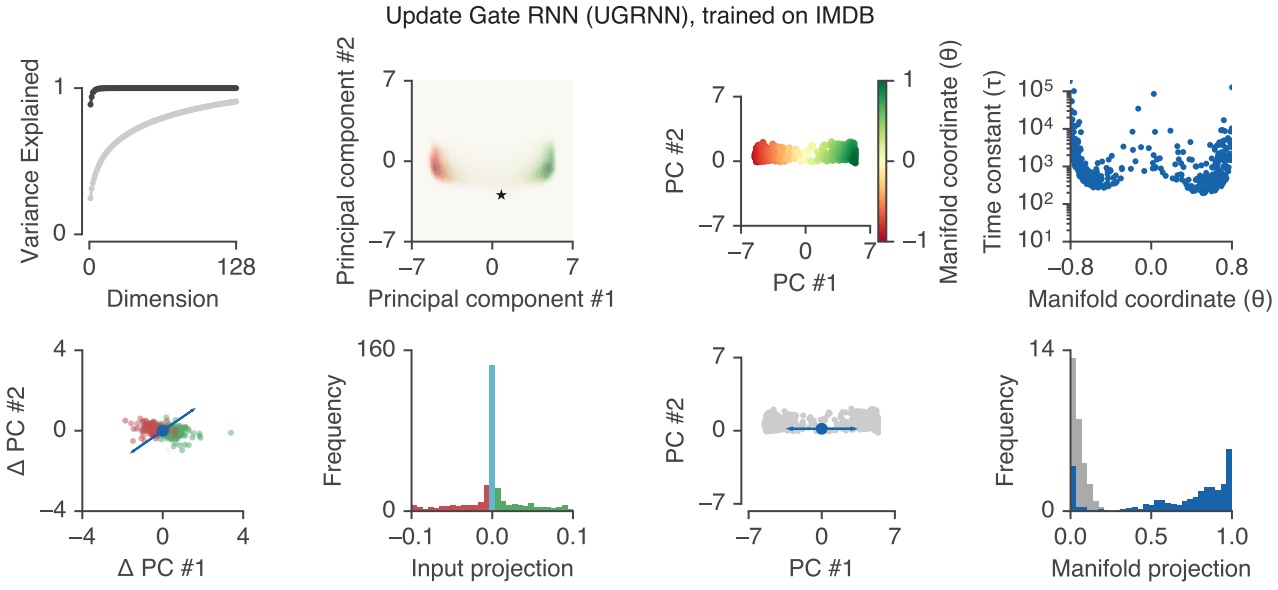

Figure 10. Summary plots for UGRNN on IMDB reviews. See first supplemental figure (LSTM on Yelp) for description.

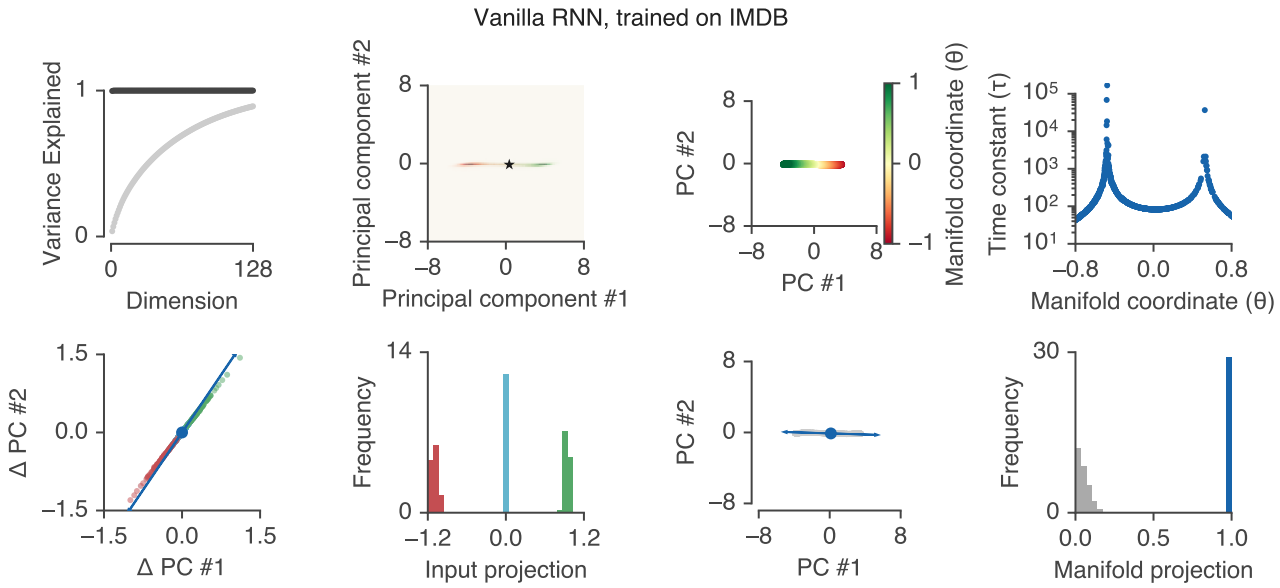

Figure 11. Summary plots for VRNN on IMDB reviews. See first supplemental figure (LSTM on Yelp) for description.

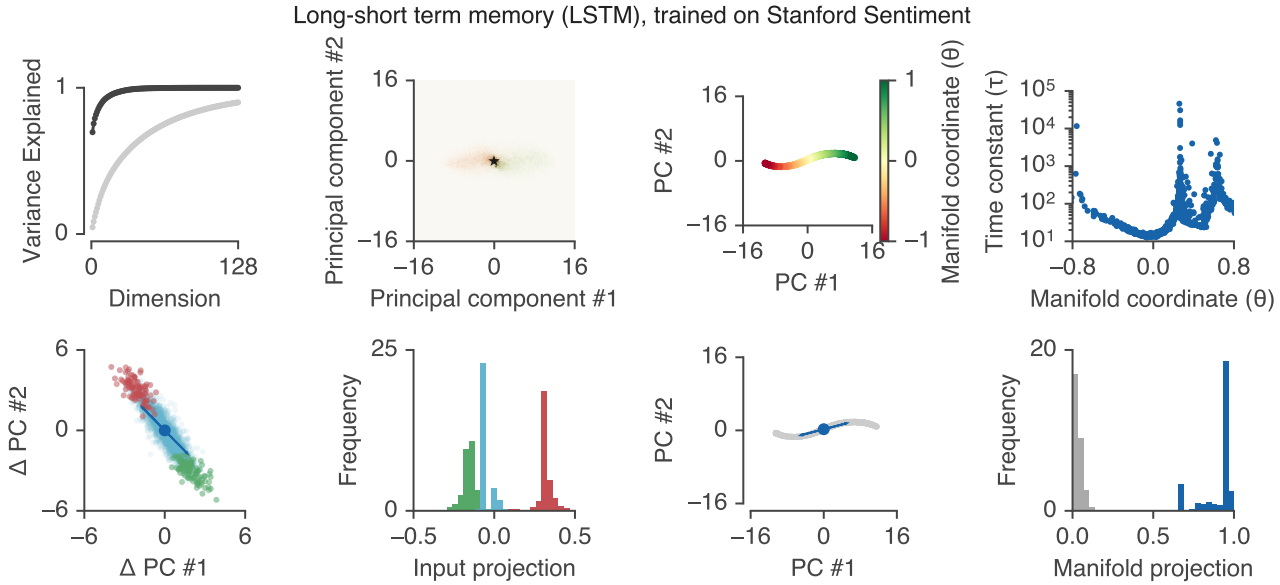

Figure 12. Summary plots for LSTM on SST reviews. See first supplemental figure (LSTM on Yelp) for description.

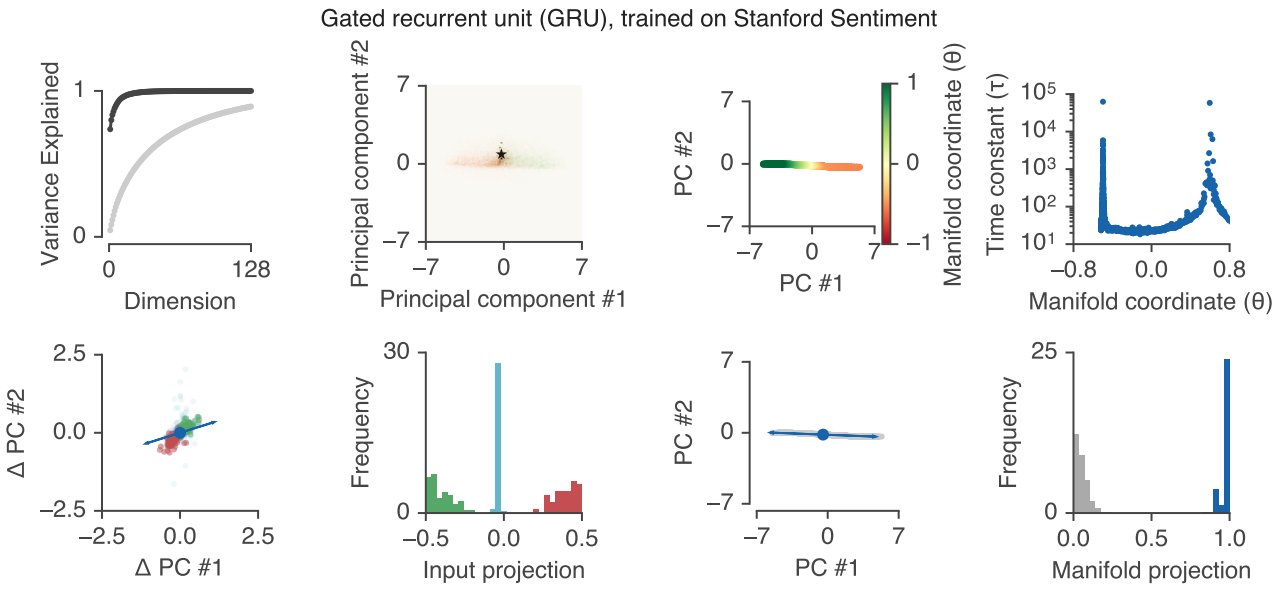

Figure 13. Summary plots for GRU on SST reviews. See first supplemental figure (LSTM on Yelp) for description.

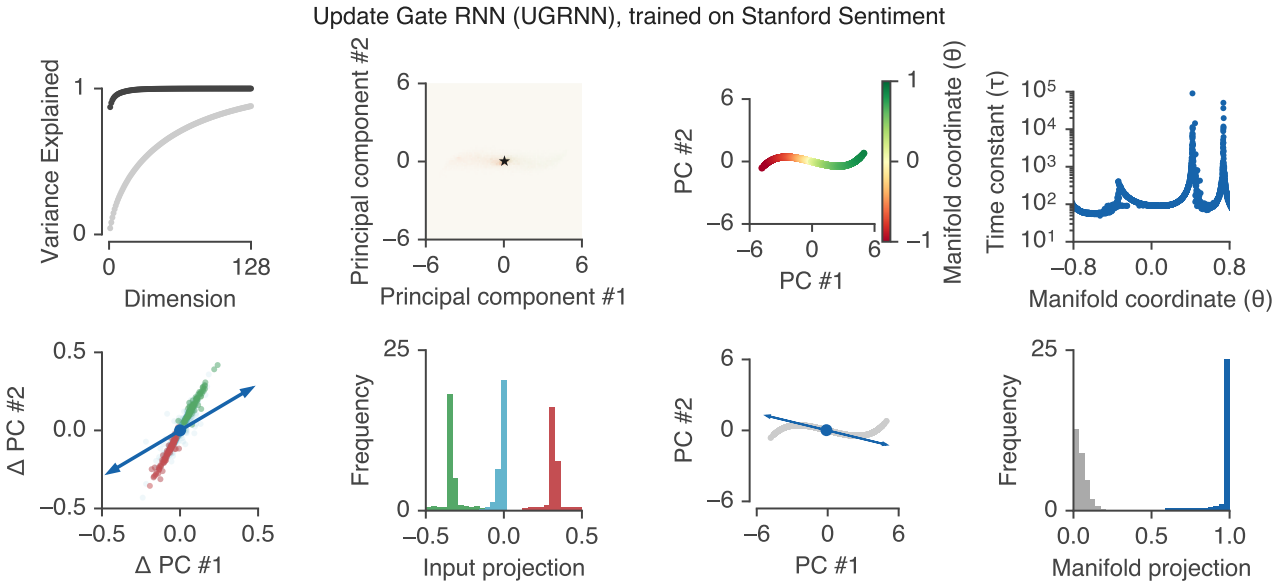

Figure 14. Summary plots for UGRNN on SST reviews. See first supplemental figure (LSTM on Yelp) for description.

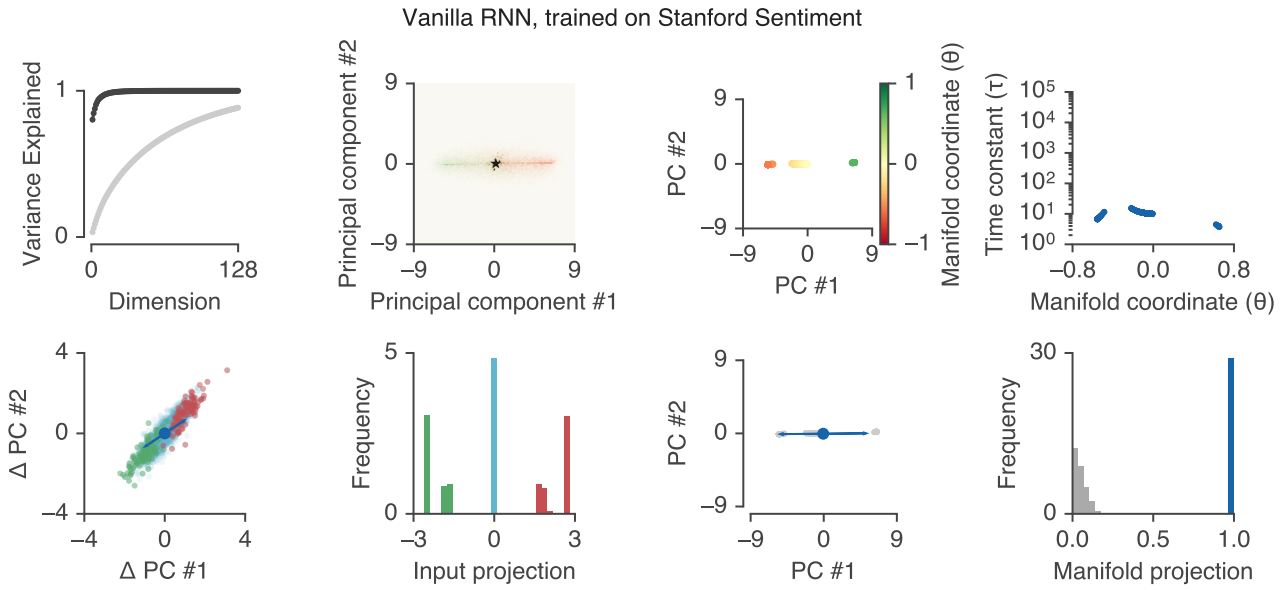

Figure 15. Summary plots for VRNN on SST reviews. See first supplemental figure (LSTM on Yelp) for description.